한국산업정보학회논문지 제7권 제5호 (2002. 12)

클래스의 응집도와 결합도를 이용한 객체 지향 설계 재구조화

(Restructuring of Object-Oriented Designs using Cohesion and Coupling of Class)

이 종 석*, 천 은 홍*

(Jong-Seok Lee, Eun-Hong Cheon)

요 약 최근의 많은 소프트웨어들은 규모가 방대하고 복잡하여 개발자가 도구를 사용하지 않고 수작업으로 설계를 재구조화하기 매우 어려우며 또한 많은 시간과 노력의 요구된다. 본 논문에서는 클래스의 응집도와 결합도를 이용하여 객체 지향 설계를 자동으로 재구조화하는 방법을 기술한다. 먼저 메소드들의 연관 관계를 규정하는 행렬을 응집도와 결합도를 이용한 메소드 사이의 거리로 정의한 후, 분리 클래스를 먼저 분할하고, 다음에 가장 낮은 응집도를 가진 클래스부터 분할하여 가장 거리가 가까운 클래스와 결합시킴으로써 설계를 재구조화한다.

Abstract Recent commercial systems are too large and too complex to restructure their design by hand without tools and cost too much time and effort. This thesis presents a restructuring approach using cohesion and coupling of classes to change object-oriented designs automatically. The matrix which represents the relation between methods is defined by distance between methods using cohesion and coupling, Class restructuring starts to isolate to dividing class, selects the class to have the lowest cohesion and combines the selected class with the closest class.

1. 서론

재구조화(restructuring)는 상대적으로 같은 추상 수준(abstraction level)에서 하나의 표현 형태로부터 다른 형태로 바꾸는 것이다. 따라서 대상 시스템의 기능이나 외적인 동작이 바뀌지 않는다. 재구조화는 전통적인 의미의 구조를 향상시키기 위하여 코드를 변형하는 것이다. 비구조적인 복잡한 프로그램을 구조적인 프로그램으로 바꾸는 것이 가장 대표적인 예이다. 그러므로 재구조화는 데이터의 이름과 정의, 프로그램의 논리 구조를 표준화하여 소프트웨어의 유지 보수성과 생산성을 높이는 작업이라고 할 수 있다.

객체 지향 설계 재구조화는 설계 구조를 모듈화하는 과정으로서 이를 위해서는 메소드와 속성들 사이의 관계를 파악하고 기능을 추론하여야 한다, 이를 위해서는 초기 설계를 추상화 모델로 표시하고 이 모델을 메트릭을 사용하여 정확하게 정량화하여 측정하여야 한다[1].

본 논문에서는 객체 지향 소프트웨어를 평가하는 메트릭 중 응집도와 결합도를 통하여 생성된 메소드들간의 연관 관계를 나타내는 행렬을 이용하여 클래스들을 분할 및 결합함으로써 객체 지향 설계를 재구조화하는 방법을 제안한다.

본 논문의 구조는 다음과 같다. 2장에서 객체 지향 설계에 대한 기존 연구를 기술한다. 3장에서는 객체 지향 설계 재구조화 과정을 기술하고, 4장에서 실제 데이터에 이를 적용한 결과를 기술한다. 5장에서는 결론과 향후 연구 계획을 기술한다.

* 우석대학교 컴퓨터공학과

2. 관련 연구

Choi와 Scachi는 모듈들 간에 자원을 교환하는 것을 측정한 후, 시스템 재구조화 알고리즘을 이용하여 계층적 설계 기술을 유도하였다. 먼저 원시 코드를 분석하여 MIL(Module Interconnection Language)로 쓰여진 설계 명세서를 생성한다. 재구조화 알고리즘은 모듈의 결합도가 최소화되도록 RFD(Resource-Flow Diagram)에서 RSD (Resource Structure Diagram)로 변형시키는 것이다[3].

Abreu와 Pereira는 객체 지향 시스템에서의 모듈화 정도를 수량화시킨 접근 방법을 이용하였다. 이 방법은 클래스를 그룹화시키기 위해 통계적 방법을 사용한 군집(cluster) 분석을 이용하였다. 군집 분석에 있어서 클래스들 사이의 상이점은 분류법에 따라 분류한 그들 간에 상대적 결합에 기반을 두고 있다. 여기서 분류된 카테고리는 가중치가 주어진다. 이 방법은 계층적 군집 방법을 사용하였기 때문에 사용할 때에는 모듈의 개수를 미리 정해야만 한다[4].

Tesch와 Klein은 CAPO(Computer Assisted Process Organization)를 제안하였는데, 이것은 시스템으로부터 계층도(hierarchy chart)를 생성하는 것이다. 프로세스들 사이의 응집도와 결합도는 자료 흐름도와 CAPO에 있는 시스템 사전으로부터 유도된다. 응집도는 선행 메트릭스(precedence matrix)와 타이밍 관계 메트릭스(timing relation matrix), 발생률 메트릭스(incidence matrix)로부터 계산되고, 결합도는 두 모듈에 의하여 공유되는 데이터의 백분율에 의해 유도된다. CAPO는 응집도와 결합도를 사용한 composite measure를 이를 최소화하도록 프로그램에 적용한다[5].

Kim과 Kwon은 출력에 종속되어 있는 함수로부터 processing block을 유도하였다. 이 processing block은 출력 변수와 관계 있는 데이터나 제어를 가진 데이터 토큰의 모임이다. 모듈은 모듈 강도에 의해 분리되거나 그룹화 된다. 모듈 강도를 나타내는 척도는 processing block들 사이의 관계 정도의 평균으로 계산된다[6].

Kang과 Beiman은 소프트웨어 설계 모델을 가지고 소프트웨어 재구조화를 위한 틀과 재구조화 기준에 기반을 둔 척도, 그리고 재구조화 과정을 제안하였다. 재구조화 모델은 모듈 컴퍼넌트와 그들 간에 연결을 보여 주기 위해 IODGs (Input-Output Dependence Graphs)와 MIGs(Module Interconnection Graph)를 사용한다. 틀은 재구조화 기준으로 DLC(Design Level Cohesion)와 DCP(Design level coupling)을 사용한다. DLC 척도는 모듈 출력쌍 사이의 여섯 개의 관계에 기반을 두고 있고, DCP 척도는 가장 강한 것에서부터 가장 낮은 것까지 정렬된 다섯 개의 관계에 의하여 분류된다. 재구조화 과정은 DLC를 증가시키고, DCP를 감소시키는 합성(composition)과 분해(decomposition) 과정의 연속으로 이루어진다[7].

3. 재구조화 연구

클래스의 응집도는 모듈 구성 요소들이 얼마나 연관되어 있는지를 측정하는 척도이고, 결합도는 모듈이 다른 모듈과 얼마나 밀접하게 연관되어 있는지를 측정하는 척도이다. 따라서 클래스의 응집도와 결합도를 측정하기 위해 객체 지향 시스템의 메소드와 메소드, 메소드와 애트리뷰트의 연관 관계를 보여주는 요소 참조 그래프 ERG(Element Reference Graph)를 정의한다. C++ 프로그래밍 언어의 생성자(constructor)는 클래스의 객체가 생성될 때 애트리뷰트에 대한 필요한 초기화만을 수행하면 되고, 소멸자(destructor)는 객체가 소멸될 때 필요한 작업만을 수행하면 된다. 따라서 이러한 생성자와 소멸자는 메소드에서 제외한다.

정의 1. ERG

객체 지향 시스템의 요소 참조 그래프 ERG = (V, E) 는 방향성 그래프로 다음과 같이 정의한다.

$V = V_m \cup V_a$

V_m : 메소드의 집합

V_a : 애트리뷰트의 집합

$E = E_m \cup E_a$

E_m : 메소드가 다른 메소드를 호출

E_a : 메소드가 애트리뷰트를 참조

각 에지의 weight는 메소드가 다른 메소드나 애트리뷰트를 참조한 횟수이다.

[그림 1]은 C++ 프로그래밍 언어와 비슷하게 쓰여진 객체 지향 시스템의 간단한 예이다. 이것을 ERG로 표현하면 [그림 2]와 같다. 그래프에서 사각형은 메소드를 나타내고 원은 애트리뷰트를 나타낸다. 또한 실선은 메소드가 애트리뷰트를 참조하는 것을 나타내고, 점선은 메소드가 다른 메소드를 호출하는 것을 나타낸다.

```
class P {
    int a2, a3;
    R r;
  public:
    int a1;
    int m1(int x) { reference a1, r.a7 }
    int m2(int x, int y) { reference a2, a2, a3 }
    double m3(void) { reference a2, a3; call m2(x, y) }
}
class Q {
    double a4, a5, a6;
    P p;
  public:
    int m4(int x) { reference a4, a5; call p.m1(x) }
    double m5(double x) { reference a4, a5, a6 }
    double m6(int x, double y) { reference a5, a6; call p.m3( ) }
}
class R {
    int a8;
    P p;
  public:
    int a7;
    int m7(Q q) { reference a7; call p.m1(x), q.m6(x, y), q.m6(x, y) }
    float m8(int x, int y) { reference a7, a8 }
    float m9(float x) { reference a8 }
}
class S: public R {
    int a9, a10;
    Q q;
  public:
    double a11;
    int m10(int x) { use a9, a11; call q.m5(x) }
    double m11(double y) { use a10, a11, r.a8 }
}
class T {
    int a12, a13;
    S s;
  public:
    int m12(int x) { use a12, a13, s.a11, s.a11; call s.m11(y) }
    int m13(double y) { use a13 }
}
```

그림 1. 예제 시스템

그래프 ERG를 통하여 객체 지향 시스템의 메소드와 메소드, 메소드와 애트리뷰트의 연관 관계를 알 수 있지만, 이를 자동화하기 쉽게 하기 위해서는 행렬로 표현하는 것이 좋다. 그러므로 메소드가 다른 메소드를 호출하는 관계를 나타내는 행렬을 MIM(Method Invoking Matrix)라고 정의한다.

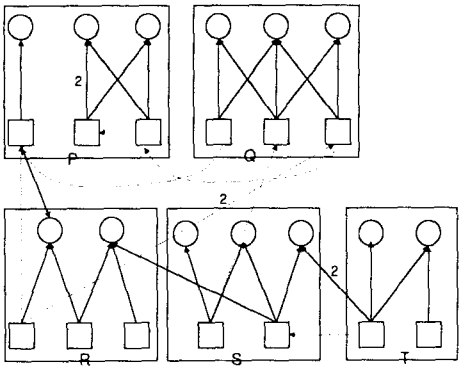

그림 2. 예제 시스템의 ERG

정의 2. MIM

행렬 MIM은 메소드가 다른 메소드를 호출하는 관계를 나타내는 행렬로 행과 열의 수는 모두 전체 시스템의 메소드의 수이다. 또한 행렬의 각 원소의 값은 메소드가 다른 메소드를 호출한 횟수이다.

0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	2	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0

그림 3. 예제 시스템의 MIM

[그림 2]에 있는 ERG에 의한 MIM은 [그림 3]과 같다.

재구조화를 위해서는 메소드들의 연관 관계를 알아야 한다. 거리 개념으로 변환하여야 한다. 이를 위해 메소드 거리 행렬(Method Distance

Matrix) MDM을 정의한다. 이 행렬은 두 메소드의 관계를 거리로 표현하기 위하여 두 메소드의 연관 관계의 역수로 나타낸다. 따라서 두 메소드의 관계가 깊으면 0에 가까운 값을 가져 메소드의 거리가 가까운 것으로 나타나고, 관계가 적으면 1에 가까운 값을 가져 거리가 먼 것으로 나타난다. 그러나 이렇게 하면 값이 0에서 1까지 분포되어 너무 좁은 범위이므로, 그 값에 100을 곱한다. 따라서 관계가 있는 메소드는 100 이하의 값을 가지게 된다. 그리고 두 메소드가 관계가 없을 경우에는 값을 200으로 하여 먼 거리임을 표시한다.

정의 3. MDM

행렬 MDM은 메소드들 간의 거리를 나타내는 행렬로 행과 열의 수는 모두 전체 시스템의 메소드의 수이다. 그리고 행렬의 각 원소의 값은 두 메소드의 연관 관계의 역수로 다음과 같이 나타낼 수 있다.

$$MDM(i,j)=\begin{cases}0, \text{ if } i=j\\ 200, \text{if } MIM(i,j)=0, MIM(j,i)=0, R_a(i)\cap R_a(j)=\varnothing\\ \dfrac{100}{\sum_{n=0}^{MIM(i,j)-1}0.5^n+\sum_{n=0}^{MIM(j,i)-1}0.5^n+cref(i,j)}, otherwise\end{cases}$$

$R_a(i)$: 메소드 i가 참조하는 애트리뷰트의 집합

$cref(i,j)$: 메소드 i와 j가 공통으로 참조하는 애트리뷰트의 수 단, 메소드 i와 j가 공통으로 참조하는 애트리뷰트 중에서 동일한 애트리뷰트를 참조할 경우 0.5를 곱함

예제 시스템의 MDM을 계산한 결과는 [그림 4]와 같다.

0	200	200	100	200	200	50	200	200	200	200	200	200
200	0	33.33	200	200	200	200	200	200	200	200	200	200
200	33.33	0	200	200	100	200	200	200	200	200	200	200
57.14	200	200	0	50	100	200	200	200	200	200	200	200
200	200	200	50	0	50	200	200	200	100	200	200	200
200	200	100	200	50	0	66.67	200	200	200	200	200	200
50	200	200	200	200	66.67	0	100	200	200	200	200	200
200	200	200	200	200	200	100	0	100	200	200	200	200
200	200	200	200	200	200	200	100	0	200	100	200	200
200	200	200	200	100	200	200	200	200	0	100	200	200
200	200	200	200	200	200	200	200	200	100	0	50	200
200	200	200	200	200	200	200	200	200	200	50	0	100
200	200	200	200	200	200	200	200	200	200	200	100	0

그림 4. 예제 시스템의 MDM

이렇게 구해진 행렬 MDM은 군집 분석에서의 거리 조건

- $d_{ij} \geq 0$, 만약 i = j 이면 $d_{ij}=0$
- $d_{ij} = d_{ji}$
- $d_{ij} + d_{jk} \geq d_{ik}$

을 만족시키므로 군집 분석을 사용하여 클래스 재구조화를 할 수 있다.

그런데 설계가 잘 되어 있는 시스템은 높은 응집도와 낮은 결합도를 갖는 시스템이다. 그러므로 클래스 재구조화는 응집도가 높고 결합도가 낮은 클래스를 만드는 과정이라고 할 수 있다. 이에 본 논문에서는 응집도와 결합도를 [2]에서 제안한 LCOM*와 CLC를 이용하여 클래스 재구조화를 하고자 한다.

클래스 재구조화 과정은 다음과 같다.

(1) 분리 클래스를 분리되지 않은 클래스로 분할한다.

분리 클래스는 가장 응집도가 낮은 클래스이다. 그러므로 이 클래스를 가장 먼저 분할하여 분리되지 않은 클래스로 나눈다. 그리고 분리된 클래스 각각의 응집도를 계산한다. 그런데 이를 위해서는 시스템에 있는 모든 클래스의 애트리뷰트와 메소드 집합을 알아야 하고, 이 집합들은 클래스가 분할 또는 병합될 때마다 변경되어야 한다. 이처럼 분리된 클래스를 분할하여 여러 개의 분리되지 않은 클래스로 만들기 때문에 재구조화를 하게 되면 분리되어 있는 클래스의 개수만큼 클래스의 수가 증가하게 된다.

(2) 응집도가 가장 낮은 클래스를 분할한다.

시스템 내에 분리 클래스가 없으면 이제 가장 응집도가 낮은 클래스를 선택하여 이 클래스 중 가장 낮은 메소드 연관도를 가진 메소드를 분리한다. 같은 응집도를 가진 클래스가 여러 개인 경우에는 임의로 한 클래스를 선택하여 분할한다. 이렇게 함으로써 클래스 내에서 응집도를 떨어뜨리는 메소드를 분리하여 클래스의 응집도를 증가시킬 수 있다. 클래스를 분할할 때 분할되는 메소드가 참조하는 애트리뷰트를 다른 메소드에서도 참조한다면 이 애트리뷰트는 양쪽 모두에 있어야 하므로 애트리뷰트의 수가 증가한다.

(3) 가장 가까운 거리에 있는 클래스와 결합한다.

각 클래스는 메소드라는 개체가 모인 군집이라고 할 수 있으므로, 가장 가까운 거리에 있는 클래스들을 결합하여 새로운 클래스를 만들 수 있다. 가장 가까운 거리에 있는 클래스를 찾기 위해서는 군집 분석 중에서 분산이 작은 군집을 형성시키는 평균 연결법을 사용한다. 평균 연결법이란 두 군집

에 있는 모든 개체들 간에 가능한 거리의 평균값이 가장 작은 군집을 합치는 방법이므로, 두 클래스에 속하는 메소드 중에서 관계를 가진 메소드 사이의 거리의 평균값으로 하여 그 값이 가장 작은 클래스의 쌍을 찾는다. 이때 같은 거리에 있는 클래스의 쌍이 여러 개인 경우에는, 임의로 한 클래스 쌍을 선택하여 결합한다면 낮은 응집도를 가진 클래스가 만들어질 수 있으므로, 결합하였을 때 높은 응집도를 가진 클래스 쌍을 선택하여 두 클래스를 결합한다.

(4) 결합된 클래스가 가장 낮은 응집도를 가질 때 종료한다.

방금 결합된 클래스가 가장 낮은 응집도를 가져 다시 분할되어야 할 때 클래스 재구조화 과정을 종료한다. 종료 시 클래스의 응집도 값 원래 시스템의 응집도 값의 평균보다 크다면 그 클래스를 분할하여 결합하기 전과 같이 만든 후 종료한다. 가장 낮은 응집도를 가진 클래스가 방금 결합된 클래스가 아니면 (2)번 과정으로 간다.

[그림 1]의 시스템의 재구조화 과정을 거친 결과는 [그림 5]와 같다. 이 시스템은 분리 클래스 P가 분할되어 클래스 R의 메소드 m7과 결합된다. 그리고 클래스가 분할될 때 메소드에 의해 참조되는 애트리뷰트는 분할된 모든 클래스 내에서 참조되어야 하기 때문에 그 수가 증가하였다.

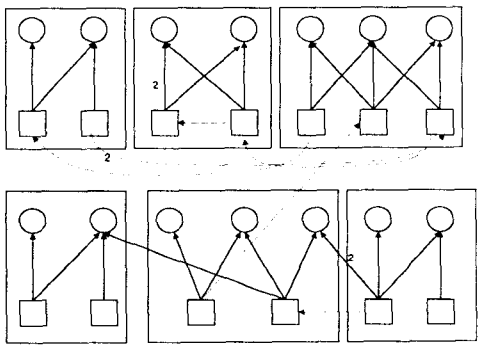

그림 5. 재구조화된 시스템의 ERG

그리고 초기 시스템에서의 응집도, 결합도와 재구조화된 시스템에서의 응집도, 결합도의 평균값은 [표 1]과 같다.

[표 1]에서와 같이 재구조화된 시스템의 응집도는 증가하였고, 결합도는 감소하였다.

[표 1] 기존 시스템과 재구조화된 시스템의 비교

	LCOM	CBO
초기 시스템	0.2	2
재구조화된 시스템	0	1.67

4. 데이터 분석

본 논문에서 제안한 클래스 재구조화 알고리즘을 분석하기 위해 C++ 프로그래밍 언어로 작성된 프로그램을 이용하여 테스트하였다. 이 프로그램은 객체 지향 설계를 위한 개발 환경 시스템(development environment system for object-oriented design : DOOD)으로, 이 시스템의 크기는 [표 2]와 같다. 재구조화 알고리즘을 DOOD 시스템에 적용한 결과는 [표 3]과 같다.

[표 2] DOOD 시스템의 크기

		줄	클래스	메소드	애트리뷰트
전체		26184	133	677	1489
클래스	Max	1549	-	140	38
	Min	56	-	1	0

[표 3] 재구조화된 DOOD 시스템

	클래스	메소드	애트리뷰트
DOOD 시스템	134	677	1489
재구조화된 시스템	187	677	1622

재구조화된 시스템은 분리 클래스가 분할되어 분리되지 않은 클래스로 만들어질 때 클래스의 개수가 증가하므로, 기존 시스템보다 클래스의 개수가 증가하여 전체 클래스의 개수는 187개가 되었다. 그리고 애트리뷰트의 수가 증가한 이유는 앞의

예제 시스템에서와 같이 클래스가 분할될 때 메소드에 의해 참조되는 애트리뷰트는 분할된 모든 클래스 내에서 참조되어야 하기 때문이다. 물론 클래스가 분할될 때 메소드는 어느 한 클래스에만 속하게 되므로 메소드의 수는 변화가 없다.

DOOD 시스템를 재구조화 결과 응집도가 향상되었는지를 살펴보기 위해, [8]에서 제안한 LCOM과 [2]에서 제안한 LCOM*에 대한 초기 상태와 재구조화된 상태에 대한 분포도가 [그림 6]과 [그림 7]에 나타나 있다.

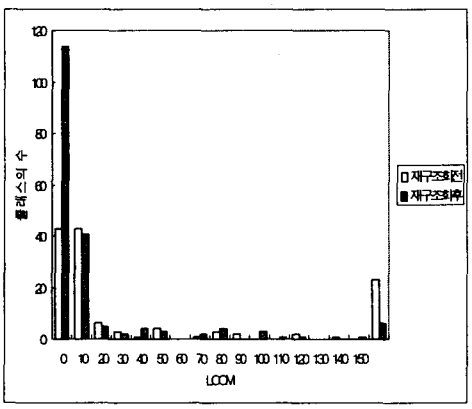

그림 6. 초기 상태와 재구조화된 후의 LCOM 값 비교

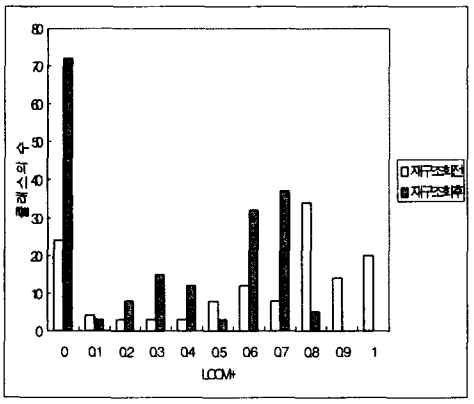

그림 7. 초기 상태와 재구조화된 후의 LCOM* 값 비교

[그림 6]에서 초기 DOOD 시스템은 LCOM* 값이 0인 클래스의 수가 134개 중 43개인데 비해, 재구조화된 시스템은 전체 187개 클래스 중에서 반이 넘는 114개의 LCOM 값이 0인 것으로 나타난다. 그리고 150보다 큰 LCOM 값을 가진 클래스의 수가 23개에서 6개로 줄었다. 그러므로 재구조화함으로써 시스템의 LCOM 값이 크게 감소하였음을 알 수 있다.

[그림 7]에서 재구조화된 시스템의 클래스 중 많은 수가 0.1보다 작은 LCOM* 값을 가진다. 재구조화되기 전에는 0.1보다 작은 LCOM* 값을 가진 클래스의 수가 25개였는데, 재구조화된 후에는 그 수가 70개를 넘었다. 이러한 이유는 DOOD 시스템에서 분리 클래스가 분할될 경우, 이 분할된 클래스들은 굉장히 높은 응집도 값을 가지기 때문이다.

DOOD 시스템를 재구조화 결과 결합도가 향상되었는지를 살펴보기 위해, [8]에서 제안한 CBO과 [2]에서 제안한 CLC에 대한 초기 상태와 재구조화된 상태에 대한 분포도가 [그림 8]과 [그림 9]에 나타나 있다.

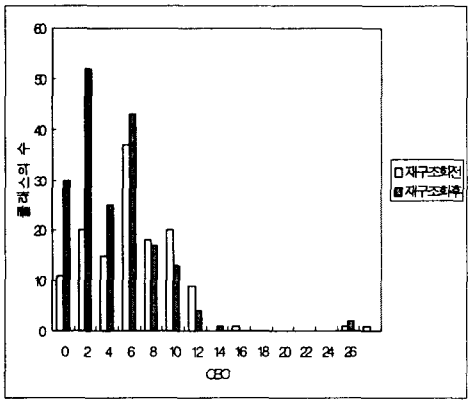

그림 8. 초기 상태와 재구조화된 후의 CBO 값 비교

[그림 8]에서 재구조화되기 전의 DOOD 시스템은 CBO 값이 4보다 작은 클래스의 수가 134개 중 31개인데 비해, 재구조화된 시스템은 전체 187개 클래스 중에서 거의 반에 가까운 82개의 CBO 값이 4보다 작은 것으로 나타난다. 그리고 전체 클래스의 수가 증가하였음에도 불구하고 10보다 큰 CBO 값을 가진 클래스의 수가 32개에서 6개로 줄었다. 그러므로 재구조화함으로써 시스템의 CBO 값이 감소하였음을 알 수 있다.

[그림 9]를 보면 60%에 가까운 클래스가 10보다 작은 CLC 값을 가진다. 이는 재구조화되기 전의

시스템에서 10보다 큰 CLC 값을 가진 클래스의 수가 전체 클래스의 60% 이상이었던 것에 비하면 결합도가 많이 낮아졌다고 할 수 있다. 또한 가장 큰 CLC 값도 60에서 32로 줄었다.

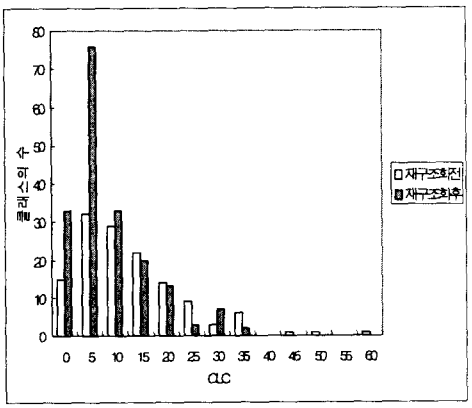

그림 9. 초기 상태와 재구조화된 후의 CLC 값 비교

DOOD 시스템의 초기 상태와 재구조화된 상태에 대한 응집도, 결합도의 평균값은 [표 4]와 같다.

[표 4] DOOD 시스템과 재구조화된 시스템의 비교

	LCOM	CBO	LOCM⁺	CLC
DOOD 시스템	91.55	5.902	0.664	11.4
재구조화된 시스템	17.337	3.968	0.322	6.963

5. 결론

시스템의 모든 메소드 사이의 관계를 표현하는 행렬 MDM을 정의한다. 이 행렬은 클래스 재구조화에 이용하기 위해 메소드 사이의 거리를 값으로 갖는다. 클래스 재구조화 방법은 분리 클래스를 먼저 분리한 후, 가장 낮은 응집도를 가진 클래스부터 분리하여 다른 클래스와 결합시킨다. 이러한 과정은 결합된 클래스가 다시 분리될 때까지 반복한다.

본 논문에서 제시한 재구조화 방법은 소프트웨어 설계 단계에서 모델링 도구로부터 정보를 추출하여 설계 품질을 높이기 위하여 적용될 수 있다.

향후 연구로는 본 연구를 규모가 더 큰 실제 시스템에 적용하여 실험적으로 그 유효성을 확인하는 것이다. 그리고 실제 객체 지향 시스템에 본 논문에서 제안한 재구조화 방법을 적용하여 클래스가 재구성된 시스템과, 전문가가 실제로 재구조화한 시스템을 비교하여 재구조화 방법의 타당성을 입증하는 연구도 필요하다.

참 고 문 헌

[1] 이병정, 우치수, "메트릭을 이용한 객체 지향 설계 재구조화", 정보과학회논문지, 소프트웨어 및 응용 제 28권 제 6호. pp414-427, 2001.

[2] 이종석, 우치수, "객체 지향 시스템에서의 클래스 응집도와 결합도 메트릭", 정보과학회논문지, 소프트웨어 및 응용 제 27권 제 6호. pp595-606, 2000.

[3] S. C. Choi and W. Scacchi, "Extracting and Restructuring the Design of Large Systems", IEEE Software, 7(1), pp. 66-71, 1990.

[4] F. B. Abreu, G. Pereira and P. Sousa, "A Coupling-Guided Cluster Analysis Approach to Reengineer the Modularity of Object-Oriented Systems", In Proceedings of 4th European Conference on Software Maintenance and Reengineering, 2000.

[5] D. Tesch and G. Klein, "Optimal Module Clustering in Program Organization", In Proceedings of the Twenty-Fourth Annual Hawaii International Conference on System Sciences, Vol. 2, pp. 238-245, 1991.

[6] H. S. Kim, Y. R. Kwon, and I. S. Chung, "Restructuring Programs through Program Slicing", International Journal of Software Engineering and Knowledge Engineering, 4(3), pp. 349-368, 1994.

[7] B. K. Kang and J. M. Beiman, "A Quantitative Framework for Software Restructuring", Journal of

Software Maintenance, 11(4), pp. 245-284, 1999.

[8] S. R. Chidamber, C. F. Kemerer, "A Metrics Suite for Object Oriented Design", IEEE Tr. on SE, Vol. 20, No. 6, pp.476-493, 1994.

이 종 석(Jong-Seok, Lee)
1988년 2월 서울대학교
계산통계학과 졸업(이학사)
1990년 8월 서울대학교 대학원
계산통계학과 졸업(이학석사)
2001년 2월 서울대학교 대학원
전기컴퓨터공학부 졸업(공학박사)
1993년 3월 ~ 현재 우석대학교 컴퓨터공학과 부교수
관심분야 : 소프트웨어 공학

천 은 홍(Eun-Hong Cheon)
1981년 2월 광운대학교
응용전자공학과 졸업(공학사)
1985년 2월 아주대학교
전자공학과 졸업(공학석사)
1998년 8월 아주대학교
컴퓨터공학과 졸업(공학박사)
1988년 9월 ~ 현재 우석대학교 컴퓨터공학과 부교수
관심분야 : 컴퓨터네트워크, 정보보안